Applications of DFS

- Cycle detection
- Topological sorting

Cycle detection:

if a back edge is present while a DFS is run on a graph, there should exist a cycle in the graph (prove yourself).

iff

if cycle then back edge

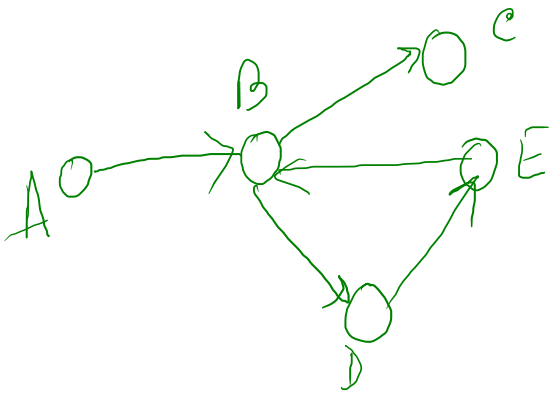
if back edge then cycle.

(u, v) → from v to u and this path is brought back to u through the back edge.

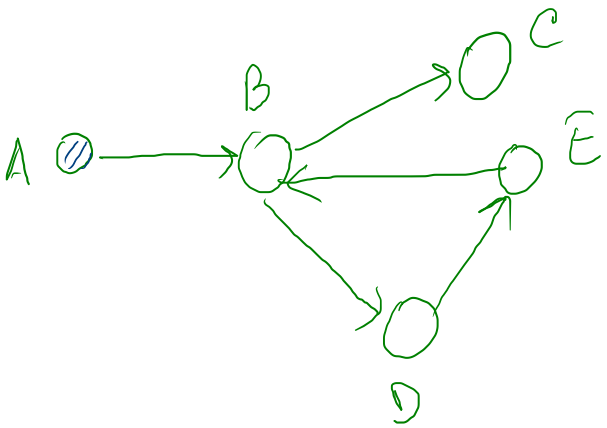
Can we modify the vanilla DFS algorithm to detect cycles?

Cycle detection:

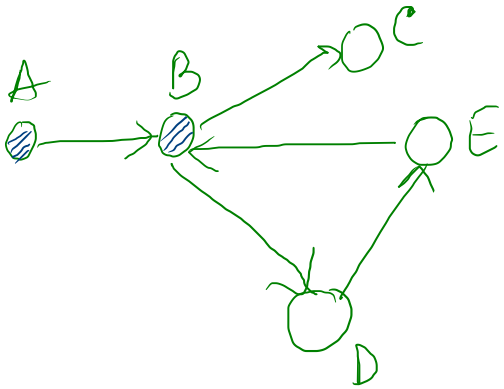
all vertices colored WHITE
create a stack S
colour v GREY and push v onto S
while S is non-empty
peek at the top u of S
if u has a GRAY neighbour, there is a cycle
else if u has a WHITE neighbour w,
color w gray and push it onto S
else color u black and pop S.



$S = \{\ \}$

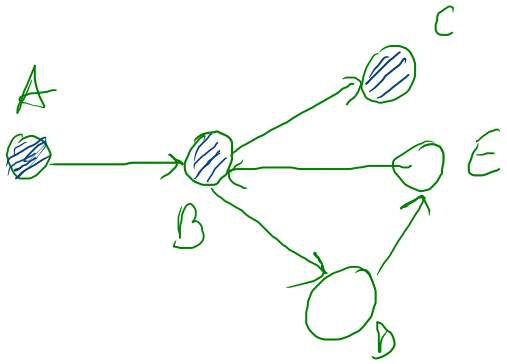


$S = A$



B

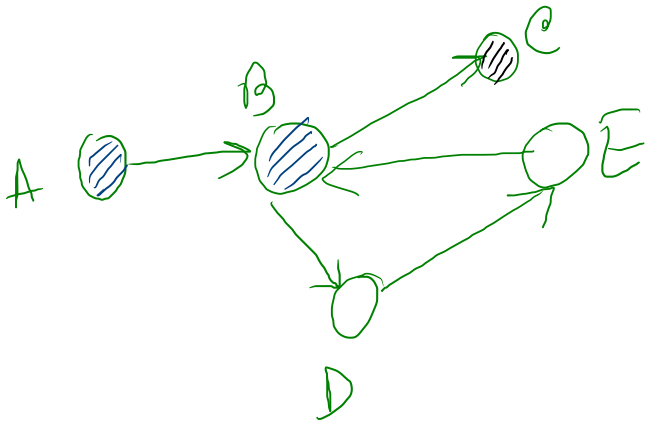
$S = A$



C

B

$S = A$



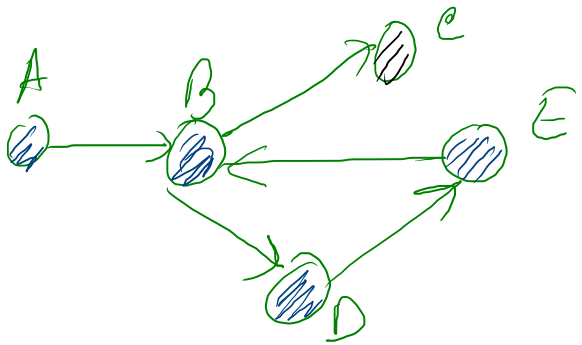
S = A

B

D

B

S = A

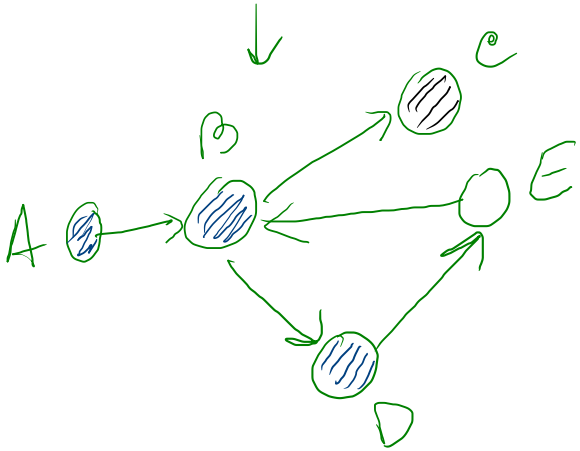


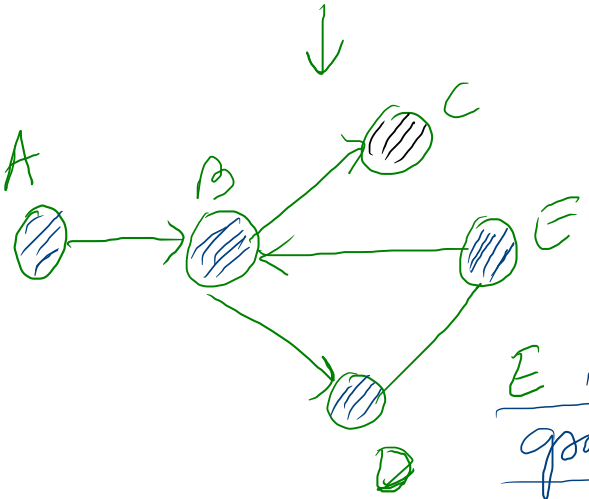
E

D

B

S = A





E has a gray nbr and that is B.

there exists a cycle in the graph.

Topological Sorting:

DAG: A DAG (directed acyclic graph) is a digraph where no path starts and ends at the same vertex (i.e. there are no cycles).

Source: A source is a node that has no incoming edges (in-degree of a source = 0)

Sink: A sink is a node that has no out going edges (out-degree = 0).

A DAG has at least one source & one sink node.

Topological sort of a DAG, $G = \langle V, E \rangle$ is a linear ordering of all its vertices such that if G_r contains an edge (u, v) then u appear before v in the ordering.

u appears before v & v appears before w in the linear ordering.

Suppose perform a set of tasks

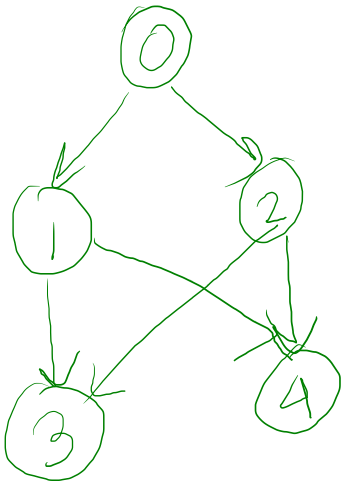
→ performed in linear order.

Only one task can be performed at a time and each task must be completed before the next task in the order begins)

↳ Topological sorting on the task dependencies

Topological sorting using DFS.

1. Perform DFS to compute the finishing times $f[v]$ for each vertex v.
2. As each vertex is finished, insert it to the front of a linked list
3. Return the linked list of vertices.



Start at 0

goto 2

goto 4

4 finished

goto 2

goto 3

3 finished

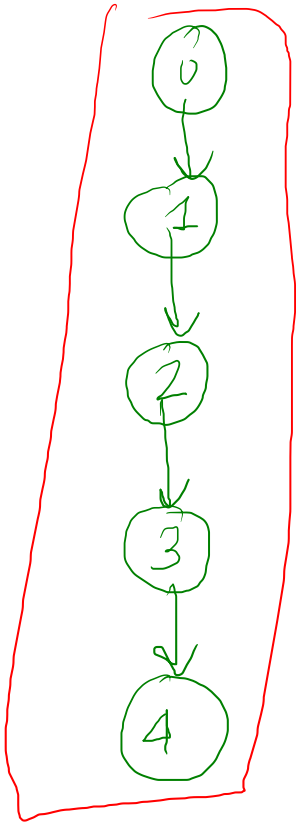
goto 2

2 finished

goto 0

goto 1 1 finished goto 0

0 finished



BFS traversal: Explores the graph through the breadth. Specifically as a layer-by-layer discovery of the vertices.

BFS(S)

color S GRAY

insert S into (initially empty) queue Q

while Q is not empty

do take vertex u off Q

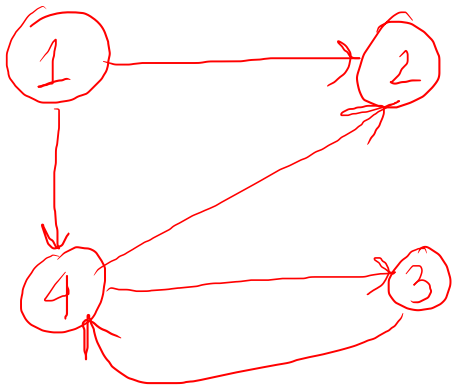
for each edge $(u, v) \in E$

do if v is WHITE // (u, v) becomes a tree edge

then color v GRAY

insert v into Q

color u BLACK.



S = 1.

2 & 4 in Q

(first 4 and then 2))

next we will process 4,

put 3 in Q.

then process 2 and then

finally 3.

layers ({1}, {2, 4}, {3})

tree edges: (1, 2), (1, 4), (4, 3).